REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jan 94	Technical

4. TITLE AND SUBTITLE
Parallel Partitioning Strategies for the Adaptive Solution of Consergation Laws

5. FUNDING NUMBERS
DAAL03-91-G-0215

6. AUTHOR(S)
Karen D. Devine, Joseph E. Flaherty, Raymond M. Loy
Stephen R. Wheat

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 29167.23-MA

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

ABSTRACT ON FIRST PAGE OF REPORT

14. SUBJECT TERMS
ON FIRST PAGE OF REPORT

15. NUMBER OF PAGES
28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Rensselaer

Department of Computer Science
Technical Report

PARALLEL PARTITIONING STRATEGIES FOR THE ADAPTIVE SOLUTION OF CONSERVATION LAWS

KAREN D. DEVINE

DEPARTMENT OF COMPUTER SCIENCE - RENSSELAER POLYTECHNIC INSTITUTE

TROY, NEW YORK 12180-3590

JOSEPH E. FLAHERTY, RAYMOND M. LOY

DEPARTMENT OF COMPUTER SCIENCE AND SCIENTIFIC COMPUTATION RESEARCH CENTER

RENSSELAER POLYTECHNIC INSTITUTE - TROY, NEW YORK 12180-3590

APPLIED MATHEMATICS AND MECHANICS SECTION

BENET LABORATORIES, WATERVLIET ARSENAL - WATERVLIET, NEW YORK 12189

AND

STEPHEN R. WHEAT

MASSIVELY PARALLEL COMPUTATION RESEARCH LABORATORY

SANDIA NATIONAL LABORATORIES - ALBUQUERQUE, NEW MEXICO 87185-1109

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

Report No. 94-1 **January 1994**

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

PARALLEL PARTITIONING STRATEGIES FOR THE ADAPTIVE SOLUTION OF CONSERVATION LAWS *

KAREN D. DEVINE†, JOSEPH E. FLAHERTY‡, RAYMOND M. LOY‡ AND STEPHEN R. WHEAT§

Abstract. We describe and examine the performance of adaptive methods for solving hyperbolic systems of conservation laws on massively parallel computers. The differential system is approximated by a discontinuous Galerkin finite element method with a hierarchical Legendre piecewise polynomial basis for the spatial discretization. Fluxes at element boundaries are computed by solving an approximate Riemann problem; a projection limiter is applied to keep the average solution monotone; time discretization is performed by Runge-Kutta integration; and a p-refinement-based error estimate is used as an enrichment indicator. Adaptive order (p-) and mesh (h-) refinement algorithms are presented and demonstrated. Using an element-based dynamic load balancing algorithm called tiling and adaptive p-refinement, parallel efficiencies of over 60% are achieved on a 1024-processor nCUBE/2 hypercube. We also demonstrate a fast, tree-based parallel partitioning strategy for three-dimensional octree-structured meshes. This method produces partition quality comparable to recursive spectral bisection at a greatly reduced cost.

Key words. Adaptive methods, hyperbolic systems of conservation laws, massively parallel computation, Galerkin finite element method, h-refinement, p-refinement, load balancing, tiling, domain decomposition, octree-derived meshes.

AMS(MOS) subject classifications. 65M20, 65M50, 65M60.

1. Introduction. Adaptive finite difference and finite element methods, which automatically refine or coarsen meshes and vary the order of accuracy of the numerical solution, offer greater robustness and computational efficiency than traditional methods. High-order methods and the combination of mesh refinement and order variation (hp-refinement) have been shown to produce effective solution techniques for elliptic [7,28] and parabolic [2,3,10,26] problems. With few exceptions [11,16], work on

* This research was supported by the U.S. Army Research Office Contract Number DAAL03-91-G-0215 and DAALO3-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center (AHPCRC) and the DoD Shared Resource Center at the AHPCRC (Flaherty, Loy); Sandia National Laboratories, operated for the U.S. Department of Energy under contract #DE-AC04-76DP00789 (Devine, Wheat), and Research Agreement AD-9585 (Devine); a DARPA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland (Loy); and the Grumman Corporate Research Center, Grumman Corporation, Bethpage, NY 11714-3580 (Loy).

† Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590.

‡ Department of Computer Science and Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590; and Applied Mathematics and Mechanics Section, Benét Laboratories, Watervliet Arsenal,Watervliet, NY 12189.

§ Massively Parallel Computation Research Laboratory, Sandia National Laboratories, Albuquerque, NM 87185-1109.

adaptive methods for hyperbolic systems has concentrated on h-refinement [5,8,12].

Distributed-memory, massively parallel computers have enabled the development of applications requiring computational resources previously unavailable. Finite difference and finite element methods for structural mechanics and fluid dynamics problems, for example, often require millions of degrees of freedom to accurately simulate physical phenomenon. When solving partial differential equations (PDEs) on MIMD computers, spatial data must be distributed across the processors' memory while minimizing the amount of data that must be exchanged between processors. This problem is especially acute when dealing with (*i*) adaptive methods, where mesh structure and work loads change during the computation, and (*ii*) three-dimensional meshes, whose data grow at a faster rate than in two-dimensions when performing h-refinement. The challenge is to combine the computational efficiency of adaptive methods with the computational resources of massively parallel computation.

We consider systems of d-dimensional hyperbolic conservation laws in m variables having the form

$$\mathbf{u}_t(\mathbf{x},t) + \sum_{i=1}^{d} \mathbf{f}_i(\mathbf{x},t,\mathbf{u})_{x_i} = 0, \quad \mathbf{x} \in \Omega, \quad t > 0, \tag{1.1a}$$

with the initial conditions

$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}^0(\mathbf{x}), \quad \mathbf{x} \in \Omega \cup \partial\Omega, \tag{1.1b}$$

and appropriate well-posed boundary conditions. The subscripts t and x_i, $i = 1,2,\ldots,d$, denote partial differentiation with respect to time and the spatial coordinates, and $\mathbf{u}$, $\mathbf{u}^0$, and $\mathbf{f}_i$, $i = 1,2,\ldots,d$, are m-vectors on the problem domain $\Omega \times (t > 0)$. Finite difference schemes for (1.1), such as the Total Variation Diminishing (TVD) [33,36] and Essentially Non-Oscillatory (ENO) [31] methods, usually achieve high-order accuracy by using a computational stencil that enlarges with order. A wide stencil makes the methods difficult to implement on unstructured meshes and limits efficient implementation on massively parallel computers. Finite element methods, however, have stencils that are invariant with method order, allowing them to easily model problems with complicated geometries and to be efficiently parallelized.

We use a discontinuous Galerkin finite element method [11,13,14] where the spatial approximation is continuous within an element, but may be discontinuous at interelement boundaries to accommodate solution discontinuities more accurately. Fluxes at element boundaries are computed by solving an approximate Riemann problem with a projection limiter applied to keep the average solution monotone near discontinuities [11,14,36]. Time discretization is performed by an explicit Runge-Kutta method.

The discontinuous Galerkin method is well suited to parallelization on massively parallel computers. The computational stencil involves only nearest-neighbor communication regardless of the degree of the piecewise polynomial approximation and the spatial dimension. Additional storage is needed for only one row of "ghost" elements along each edge of a processor's subdomain. Thus, the size of the problem scales easily with the number of processors. Indeed, for two-dimensional problems on rectangular domains with periodic boundary conditions, scaled parallel efficiencies in excess of 97% are achieved [11].

To achieve parallel efficiency with irregular structures, parallel finite element methods often use *static* load balancing [19,21] as a precursor to obtaining a finite element solution. Parallel efficiency degrades substantially due to processor load imbalances with adaptive enrichment. Even with the lower parallel efficiency, however, execution times for comparable accuracy are shorter with adaptive methods than for fixed-order methods.

We have developed an adaptive p-refinement method for two-dimensional systems that uses *dynamic* load balancing to adjust the processor decomposition in the presence of nonuniform and changing work loads. *Tiling* [37] is a modification of a dynamic load balancing technique developed by Leiss and Reddy [24] that balances work within overlapping processor neighborhoods to achieve a global load balance. Work is migrated from a processor to others within the same neighborhood. We demonstrate the improved performance obtained from a combination of p-adaptivity and parallel computation on several examples using a 1024-processor nCUBE/2 hypercube.

For three-dimensional problems with irregular grids of tetrahedral elements and adaptive h-refinement, we have developed a tree-based mesh partitioning technique that exploits the properties of tree-structured meshes. The rich, hierarchical structure of these meshes allows them to be divided into components along boundaries of the tree structure. Our partitioning technique is based on two tree traversals that (*i*) calculate the processing costs of all subtrees of a node, and (*ii*) form the partitions. Our method is inexpensive and, thus, has an advantage relative to other global partitioning techniques [21,23,27]. We demonstrate the performance of the tree-based mesh partitioning technique on a variety of three-dimensional meshes and discuss extension of the technique for parallel implementation and dynamic load balancing. We present results, using a Thinking Machines CM-5 computer, for the adaptive h-refinement solutions of an Euler flow past a cone.

2. The Discontinuous Galerkin Method. Partition the domain Ω into polygonal elements Ω_j, $j = 1, 2, \ldots, J$, and construct a weak form of the problem by multiplying (1.1a) by a test function $\mathbf{v} \in L^2(\Omega_j)$ and

integrating the result on Ω_j to obtain

$$(2.1)\quad \int_{\Omega_j} \mathbf{v}^{\mathrm{T}}\mathbf{u}_t\,d\tau + \sum_{i=1}^{d}\left[\int_{\Omega_j}(\mathbf{v}^{\mathrm{T}}\mathbf{f}_i(\mathbf{u}))_{x_i}\,d\tau - \int_{\Omega_j}\mathbf{v}_{x_i}^{\mathrm{T}}\mathbf{f}_i(\mathbf{u})\,d\tau\right] = 0.$$

Apply the Divergence Theorem to (2.1) to obtain

$$(2.2)\quad \int_{\Omega_j}\mathbf{v}^{\mathrm{T}}\mathbf{u}_t\,d\tau - \sum_{i=1}^{d}\int_{\Omega_j}\mathbf{v}_{x_i}^{\mathrm{T}}\mathbf{f}_i(\mathbf{u})\,d\tau + \sum_{i=1}^{d}\int_{\partial\Omega_j}\mathbf{v}^{\mathrm{T}}\mathbf{f}_i(\mathbf{u})n_i\,d\sigma = 0,$$

where $\mathbf{n} = [n_1, n_2, \ldots, n_d]^{\mathrm{T}}$ is the unit outward normal to $\partial\Omega_j$. Approximate $\mathbf{u}(\mathbf{x},t)$ on Ω_j by a p^{th}-degree polynomial $\mathbf{U}_j(\mathbf{x},t) \in \mathbf{S}_j \subset L^2(\Omega_j)$, and test against all functions $\mathbf{V} \in \mathbf{S}_j$. With initial conditions determined by local L^2 projection, this approximation yields the ordinary differential system

$$(2.3a)\quad \int_{\Omega_j}\mathbf{V}^{\mathrm{T}}(\mathbf{U}_j)_t\,d\tau \;-\; \sum_{i=1}^{d}\int_{\Omega_j}\mathbf{V}_{x_i}^{\mathrm{T}}\mathbf{f}_i(\mathbf{U}_j)\,d\tau + \sum_{i=1}^{d}\int_{\partial\Omega_j}\mathbf{V}^{\mathrm{T}}\mathbf{f}_i(\mathbf{U}_j)n_i\,d\sigma = 0, \quad t > 0,$$

$$(2.3b)\quad \int_{\Omega_j}\mathbf{V}^{\mathrm{T}}(\mathbf{U}^j - \mathbf{u}^0)\,d\tau = 0, \quad t = 0, \quad \forall\mathbf{V} \in \mathbf{S}_j,$$

which we solve on Ω_j, $j = 1, 2, \ldots, J$, by explicit Runge-Kutta integration of order p. Integrals are evaluated numerically using Gauss-Legendre quadrature. A basis for the local space $\mathbf{S}_j$ may be defined using products of one-dimensional Legendre polynomials, as distinct from hierarchical bases for elliptic and parabolic systems [34] which use integrals of Legendre polynomials. Two-dimensional bases involving tensor products of Legendre polynomials have been constructed for quadrilateral [11] and triangular [15] elements. A three-dimensional basis for tetrahedral elements could follow procedures developed for elliptic systems [34]. Results presented in Section 5 for tetrahedral-element meshes involve only piecewise constant approximations.

The normal component of the flux

$$(2.4)\quad \mathbf{f}_{\mathbf{n}}(\mathbf{u}) = \sum_{i=1}^{d}\mathbf{f}_i(\mathbf{u})n_i$$

remains unspecified on $\partial\Omega_j$ with (2.3) since the approximate solution is discontinuous there. We specify it using a "numerical flux" function $\mathbf{h}(\mathbf{U}_j^+, \mathbf{U}_j^-)$ dependent on solution states $\mathbf{U}_j^+$ and $\mathbf{U}_j^-$ on the inside and

outside, respectively, of $\partial\Omega_j$. Several numerical flux functions are possible [14,31]. In two dimensions [11,15], we have used the Lax-Friedrichs numerical flux,

$$\mathbf{h}(\mathbf{U}_j^+, \mathbf{U}_j^-) = \frac{1}{2}[\mathbf{f_n}(\mathbf{U}_j^+) + \mathbf{f_n}(\mathbf{U}_j^-) - |\alpha|(\mathbf{U}_j^- - \mathbf{U}_j^+)], \tag{2.5a}$$

$$\alpha = \max(|\lambda(\mathbf{f_{n_u}}(\mathbf{U}_j))|), \quad U_j^+ \le U_j \le U_j^-, \tag{2.5b}$$

where $\lambda(\mathbf{f_{n_u}})$ is an eigenvalue of the Jacobian $\mathbf{f_{n_u}}$.

In three dimensions, we use van Leer's flux vector splitting [25,35] to construct a numerical flux. This technique is not generally applicable but does apply to the Euler equations of compressible flow which have the solution and flux vectors

$$\begin{aligned} \mathbf{u}' &= [\rho, \rho u', \rho v', \rho w', e]^{\mathrm{T}}, \\ \mathbf{f}_1(\mathbf{u}') &= \left[\rho u', \rho u'^2 + p, \rho u'v', \rho u'w', u'(e+p)\right]^{\mathrm{T}}, \\ \mathbf{f}_2(\mathbf{u}') &= \left[\rho v', \rho u'v', \rho v'^2 + p, \rho v'w', v'(e+p)\right]^{\mathrm{T}}, \\ \mathbf{f}_3(\mathbf{u}') &= \left[\rho w', \rho u'w', \rho v'w', \rho w'^2 + p, w'(e+p)\right]^{\mathrm{T}}, \end{aligned} \tag{2.6}$$

where ρ, e, and p are the density, energy, and pressure; u' is the velocity component in the direction of $\mathbf{n}$; and v' and w' are velocity components tangent to $\partial\Omega_j$. The numerical flux $\mathbf{h}$ on $\partial\Omega_j$ is computed as

$$\mathbf{h}(\mathbf{U}_j'^-, \mathbf{U}_j'^+) = \mathbf{f}_1^+(\mathbf{U}_j'^+) + \mathbf{f}_1^-(\mathbf{U}_j'^-) \tag{2.7a}$$

where

$$\begin{array}{llllll} \mathbf{f}_1^+(\mathbf{U}') = \mathbf{f}_1(\mathbf{U}'), & \mathbf{f}_1^-(\mathbf{U}') = 0, & \text{if } M' \ge 1; \\ \mathbf{f}_1^+(\mathbf{U}') = 0, & \mathbf{f}_1^-(\mathbf{U}') = \mathbf{f}_1(\mathbf{U}'), & \text{if } M' \le -1; \end{array} \tag{2.7b}$$

$$\mathbf{f}_1^{\pm}(\mathbf{U}') = \pm\rho\frac{a(M'\pm 1)^2}{4}\begin{bmatrix} 1 \\ a\left[(\gamma-1)M' \pm 2\right]/\gamma \\ v' \\ w' \\ \frac{a^2\left[(\gamma-1)M'\pm 2\right]^2}{2(\gamma^2-1)} + \frac{v'^2+w'^2}{2} \end{bmatrix}, \quad \text{if } |M'| \le 1. \tag{2.7c}$$

where $M' = U'/a$, and a is the local speed of sound. Van Leer's numerical flux has the property that the two components of the normal flux, $\mathbf{f}_1^+$ and $\mathbf{f}_1^-$, depend only on the solution on the same side of $\partial\Omega_j$. Therefore, the flux may be calculated by computing each component separately, exchanging $\mathbf{f}_1^+$ and $\mathbf{f}_1^-$ between elements, and summing. This splitting approximately halves the computational and parallel communications effort relative to other flux evaluation schemes.

In regions where the numerical solution is smooth, the discontinuous Galerkin method produces the $O(h^{p+1})$, $h = \max_{i=1,2,\ldots,d,\ j=1,2,\ldots,J}(\Delta x_{i,j})$, convergence expected in, e.g., L^1 for a p^{th}-degree approximation [11,14]. To prevent spurious oscillations that develop near discontinuities with high-order methods, we have developed a projection limiter that limits solution moments [11,14,36]. Using a one-dimensional ($d = 1$) scalar problem and the Legendre polynomial basis

$$U_j(\xi, t) = \sum_{l=0}^{p} c_{jl}(t) P_l(\xi) \tag{2.8}$$

as an illustration, the coefficient c_{jk} is proportional to the k^{th} moment $\mathcal{M}_{jk}$ of U_j; i.e.,

$$\mathcal{M}_{jk} := \int_{-1}^{1} U_j(\xi, t) P_k(\xi)\, d\xi = \frac{2}{2k+1} c_{jk}, \qquad k = 0, 1, \ldots, p-1, \quad j = 1, 2, \ldots, J. \tag{2.9}$$

Thus, to keep $\mathcal{M}_{jk}$ monotone, we must keep c_{jk} monotone on neighboring elements, which we do by specifying

$$(2k+1)c_{j,k+1} =$$

$$minmod((2k+1)c_{j,k+1}, c_{j+1,k} - c_{j,k}, c_{j,k} - c_{j-1,k}), \tag{2.10a}$$

where

$$minmod(a, b, c) =$$

$$\begin{cases} \text{sign}(a) \min(|a|, |b|, |c|), & \text{if} \quad \text{sign}(a) = \text{sign}(b) = \text{sign}(c) \\ 0, & \text{otherwise.} \end{cases} \tag{2.10b}$$

The limiter (2.10) is applied adaptively. First, the highest-order coefficient c_{jp} is limited. Then the limiter is applied to successively lower-order coefficients whenever the next higher coefficient on the interval has been changed by the limiting. The higher-order coefficients are re-limited using the updated low-order coefficients when necessary. In this way, the limiting is applied only where it is needed, and accuracy is retained in smooth regions. For two- and three-dimensional problems, the one-dimensional limiter is applied in the direction $\mathbf{n}$ normal to $\partial\Omega_j$.

For vector systems, the scalar limiting is applied to the characteristic fields of the system [13]. The diagonalizing matrices $\mathbf{T}(\mathbf{u})$ and $\mathbf{T}^{-1}(\mathbf{u})$ (consisting of the right and left eigenvectors of the Jacobian $\mathbf{f}_{\mathbf{n_u}}$) are evaluated using the average values of $\mathbf{U}_j$, $j = 1, 2, \ldots, J$, on Ω_j; the scalar limiting is applied to each field of the characteristic vector; and the result is transformed back to physical space by post-multiplication by $\mathbf{T}^{-1}(\mathbf{U}_j)$.

3. Adaptive p-Refinement. We have developed an adaptive p-refinement version of the two-dimensional method (2.3, 2.5, 2.10) using rectangular grids and a method-of-lines framework. A spatial error estimate is used to control order variation procedures that attempt to keep

$$e(t) = \sum_{j=1}^{J} e_j(t) \leq \varepsilon, \tag{3.1a}$$

where ε is prescribed and

$$e_j(t) = \| \int_{\Omega_j} |\mathbf{u}(\mathbf{x},t) - \mathbf{U}_j(\mathbf{x},t)|\, d\tau \|_\infty. \tag{3.1b}$$

Control is done locally with a goal of maintaining

$$e_j(t) \leq \mathrm{TOL} = \frac{\varepsilon}{J}, \quad j = 1,2,\ldots,J. \tag{3.2}$$

We initialize $\mathbf{U}_j(\mathbf{x},0)$, $j = 1,2,\ldots,J$, to the lowest-degree polynomial satisfying (3.2) at $t = 0$. For times $t > 0$, we use p-refinement to calculate an estimate $E_j(t)$ of e_j as

$$E_j = \| \int_{\Omega_j} |\mathbf{U}_j^{p+1} - \mathbf{U}_j^{p}| d\tau \|_\infty, \quad j = 1,2,\ldots,J, \tag{3.3}$$

where $\mathbf{U}_j^p$ is the p^{th}-degree finite element approximation of $\mathbf{u}$. While this estimate is computationally expensive, it is still less expensive than h-refinement (Richardson's extrapolation) techniques and can be used to reduce the effort involved in recomputing $\mathbf{U}_j$ and its error estimate when p-refinement is needed.

A less expensive error estimation procedure similar to successful procedures for elliptic and parabolic systems [4,6] can be obtained by the $(p+1)^{st}$-degree polynomial correction to a p^{th}-degree solution while making use of superconvergence to reduce complexity. We construct a $(p+1)^{st}$-degree correction term $\mathbf{K}_j(\mathbf{x},t)$ whose roots are the superconvergence points of the approximation, and then estimate e_j as

$$E_j(t) = \| \int_{\Omega_j} |\mathbf{K}_j(\mathbf{x},t)|\, d\tau \|_\infty, \quad j = 1,2,\ldots,J. \tag{3.4a}$$

For $p = 0$, the superconvergence points remain at the Legendre roots, but for $p > 0$, the superconvergence points move toward the Radau points [1,11], i.e., the roots of

$$R_{p+1}(\xi) = \begin{cases} P_{p+1}(\xi) - P_p(\xi), & \text{if } \mathbf{f}_{\mathbf{n_u}} > 0, \\ P_{p+1}(\xi) + P_p(\xi), & \text{if } \mathbf{f}_{\mathbf{n_u}} < 0, \end{cases} \tag{3.4b}$$

as t increases. Then, for a two-dimensional approximation using a basis of tensor products of Legendre polynomials on rectangular elements,

$$\text{(3.4c)}\quad \mathbf{K}_j(\xi,\eta,t) = \begin{cases} \mathbf{c}_{j11}(t)P_1(\xi)P_1(\eta) \\ \quad +\mathbf{c}_{j10}(t)P_1(\xi)P_0(\eta) \\ \quad +\mathbf{c}_{j01}(t)P_0(\xi)P_1(\eta), & \text{if } p=0 \\ \\ \mathbf{c}_{j,p+1,p+1}(t)R_{p+1}(\xi)R_{p+1}(\eta) \\ \quad +\sum_{k=0}^{p}(\mathbf{c}_{jk,p+1}(t)P_k(\xi)R_{p+1}(\eta) \\ \qquad +\mathbf{c}_{j,p+1,k}(t)R_{p+1}(\xi)P_k(\eta)), & \text{if } p>0. \end{cases}$$

To compute $\mathbf{K}_j(\mathbf{x},t)$, let $\hat{\mathbf{U}}_j = \mathbf{U}_j + \mathbf{K}_j$, $j = 1,2,\ldots,J$, substitute $\hat{\mathbf{U}}_j$ into (2.3), and solve for the coefficients of $\mathbf{K}_j(\mathbf{x},t)$ with $\mathbf{U}_j$ fixed.

To compute E_j using (3.4), we solve $2p+3$ additional ordinary differential equations in two dimensions, compared to an additional $(p+2)^2$ differential equations required for (3.3). The movement of the superconvergence points from the Legendre points at $t=0$ toward the Radau points for $t>0$ is gradual, occurring over several time steps [11]. Thus, the effectiveness of the estimate improves as the computation progresses.

After each time step, we compute E_j, $j = 1,2,\ldots,J$, and increase the polynomial degree of $\mathbf{U}_j$ by one if $E_j > \text{TOL}$. The solution $\mathbf{U}_j$ and the error estimate are recomputed on enriched elements, and further increases of degree occur until $E_j \le \text{TOL}$ on all elements.

We reduce the need for back-tracking by predicting the degree of the approximation needed to satisfy the accuracy requirements during the next time step. After a time step is accepted, if $E_j > H_{max}\text{TOL}$, $H_{max} \in (0,1]$, we increase the degree of $\mathbf{U}_j(\mathbf{x},t+\Delta t)$ for the next time step. If $E_j < H_{min}\text{TOL}$, $H_{min} \in [0,1)$, we decrease the degree of $\mathbf{U}_j(\mathbf{x},t+\Delta t)$ for the next time step.

EXAMPLE 1. We demonstrate the accuracy of the error estimate (3.4) in terms of its effectivity index

$$\text{(3.5)}\qquad \Theta = \frac{\text{Estimated Error}}{\text{Actual Error}}$$

for the two-dimensional problem

$$\text{(3.6a)}\qquad u_t + u_x + u_y = 0, \quad -1 < x,y < 1, \quad t > 0,$$

with

$$\text{(3.6b)}\qquad u^0(x,y) = \sin(\pi x)\sin(\pi y), \quad -1 \le x,y \le 1,$$

and periodic boundary conditions. In Table 3.1, we show the actual errors and effectivity indices with $p = 0, 1,$ and 2. Each time the mesh is refined,

the time step is halved, and the number of time steps is doubled. Effectivity indices are near unity for the entire range of computation when $p = 0$. For $p = 1$ and 2, the error estimate improves as the mesh is refined since the superconvergence points move closer to the Radau points after each time step.

	Number of Elements	Actual Error	Θ
$p = 0$	16×16	$2.66838e-1$	0.967
	32×32	$1.33946e-1$	0.969
	64×64	$6.70306e-2$	0.973
	128×128	$3.35206e-2$	0.976
	256×256	$1.67605e-2$	0.978
$p = 1$	16×16	$1.45948e-2$	0.540
	32×32	$4.21090e-3$	0.805
	64×64	$1.11300e-3$	0.975
	128×128	$2.79793e-4$	1.000
	256×256	$6.99557e-5$	1.000
$p = 2$	16×16	$6.41413e-4$	0.557
	32×32	$9.68358e-5$	0.646
	64×64	$9.68224e-6$	1.128
	128×128	$1.26721e-6$	1.009
	256×256	$1.58712e-7$	1.000
	512×512	$1.98384e-8$	1.000

TABLE 3.1
Errors and effectivity indices Θ at $t = 0.025$ using (3.4) for Example 1.

EXAMPLE 2. Consider

$$u_t + 2u_x + 2u_y = 0, \quad 0 < x, y < 1, \quad t > 0, \tag{3.7a}$$

with initial and Dirichlet boundary conditions specified so that the exact solution is

$$u(x, y, t) = \frac{1}{2}(1 - \tanh(20x - 10y - 20t + 5)), \quad 0 \le x, y \le 1. \tag{3.7b}$$

In Figure 3.1, we show the exact solution of (3.7) at time $t = 0$ and the degrees generated on a adaptive 16×16-element mesh to satisfy the initial data when $\mathrm{TOL} = 10^{-5}$.

We solve (3.7) by both fixed-order and adaptive p-refinement methods on $0 < t \le 0.1$. In Figure 3.2, we show the global L^1-error versus the CPU time for fixed-order methods with $p = 0$, 1, and 2 on 8×8, 16×16, 32×32, and 64×64-element meshes, and the p-adaptive method with $H_{max} = 0.9$, $H_{min} = 0.1$, and TOL ranging from 5×10^{-9} to 5×10^{-4} on

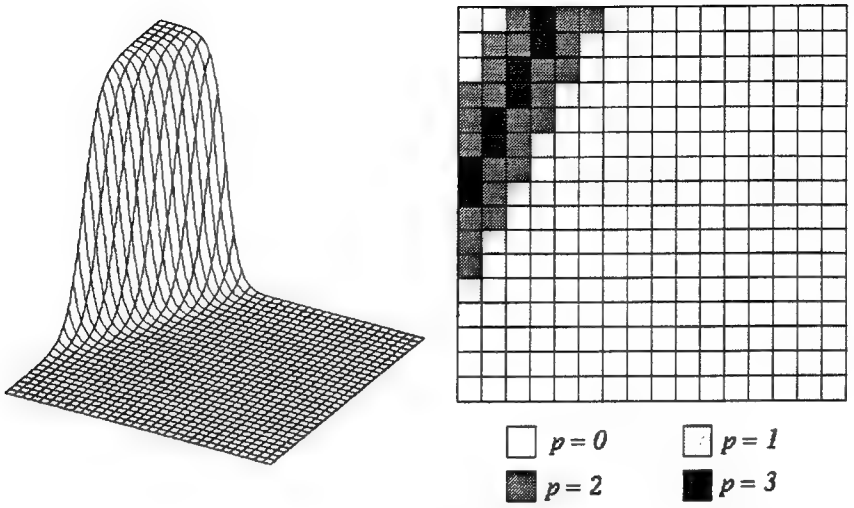


FIG. 3.1. *Exact solution of (3.7) at* $t = 0$ *and degrees generated on a* 16×16*-element mesh with* $\mathrm{TOL} = 10^{-5}$ *for Example 2.*

a 16×16-element mesh. The adaptive p-refinement method requires more computation than the fixed-order methods for large error tolerances, but because of its increasing convergence rate, it requires less work than the fixed-order methods to obtain small errors.

4. Dynamic Load Balancing via Tiling. Tiling [37,38] is a modification of the global load balancing technique of Leiss and Reddy [24,29] who used local balancing within overlapping processor neighborhoods to achieve a global load balance. A neighborhood is defined as a processor at the center of a circle of some predefined radius and all other processors within the circle. Processors within a given neighborhood are balanced with respect to each other using local performance measurements. Individual processors may belong to several neighborhoods. Work can be migrated from a processor to any other processor within the same neighborhood. In tiling, we extend the definition of a neighborhood to include all processors having finite elements that are neighbors of elements in the center processor (see Figure 4.1). Tiling neighborhoods are not related to the hardware interconnection of the processors as were the neighborhoods of Leiss and Reddy [24]. Every processor is the center of one neighborhood, and may belong to many neighborhoods. Elements are migrated only to processors having neighbors of the migrating elements.

The tiling algorithm consists of (*i*) a computation phase and (*ii*) a balancing phase, and is designed to be independent of the application. The computation phase corresponds to the application's implementation without load balancing. Each processor operates on its local data, exchanges

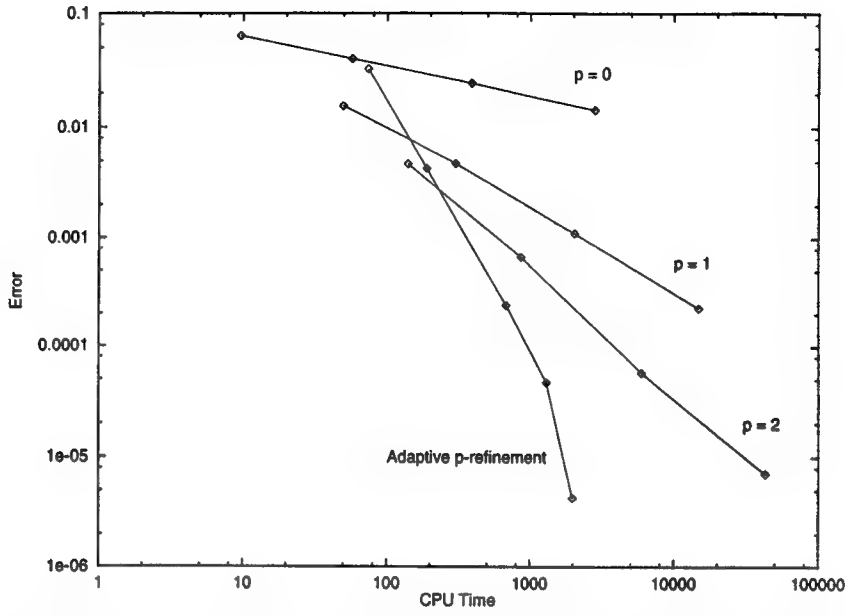


FIG. 3.2. *Convergence of the adaptive p-refinement method and fixed-order methods with* $p = 0, 1,$ and 2 *for Example 2.*

inter-processor boundary data, and processes the boundary data. A balancing phase restores load balance following a given number of computation phases. Each balancing phase consists of the following operations:

1. **Determine work loads.** Each processor determines its work load as the time to process its local data since the previous balancing phase less the time to exchange inter-processor boundary data during the computation phase. Neighborhood average work loads are also calculated.
2. **Determine processor work requests.** Each processor compares its work load to the work load of the other processors in its neighborhood and determines those processors having loads greater than its own. If any are found, it selects the one with the greatest work load (ties are broken arbitrarily) and sends a request for work to that processor. Each processor may send only one work request, but a single processor may receive several work requests.
3. **Select elements to satisfy work requests.** Each processor prioritizes the work requests it receives based on the request size, and determines which elements to export to the requesting processor. Elemental processing costs are used so that the minimum number of elements satisfying the work request are exported. (This approach differs from Wheat [37], where the average cost per element is used to determine the number of export elements). Details of the selection algorithm follow.
4. **Notify and transfer elements.** Once elements to be exported

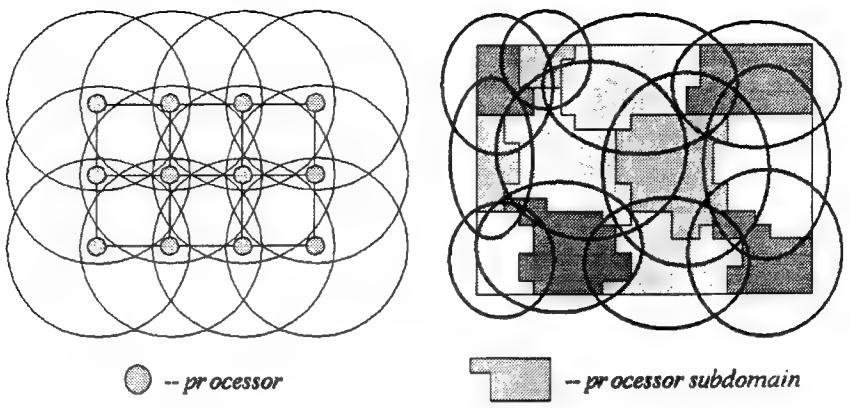


FIG. 4.1. *Examples of 12 processors in 12 neighborhoods using the Leiss/Reddy[24,29] (left) and the tiling (right) definitions.*

have been selected, the importing processors and processors containing neighbors of the exported elements are notified. Importing processors allocate space for the incoming elements, and the elements are transferred.

Each processor knows the number of computation phases to perform before entering the balancing phase. Synchronization guarantees that all processors enter the balancing phase at the same time.

The technique for selecting elements gives priority to elements with neighbors in the importing processor to prevent the creation of "narrow, deep holes" in the element structures. Elements are assigned priorities (initially zero) based upon the locality of their element neighbors. An element's priority is decreased by one for each element neighbor in its own processor, increased by two for each neighbor in the importing processor, and decreased by two for each neighbor in some other processor. Thus, elements whose neighbors are already in the importing processor are more likely to be exported to that processor than elements whose neighbors are in the exporting processor or some other processor. When an element has no neighboring elements in its local processor, it is advantageous to export it to any processor having its neighbors. Thus, "orphaned" elements are given the highest export priority. When two or more elements have the same priority, the processor selects the element with the largest work load that does not cause the exported work to exceed the work request or the work available for export.

In Figure 4.2, we illustrate an example of element priorities and selection for satisfying a work request of 55 units from the east neighboring processor. Initially, elements 3, 6, 9, and 12 are eligible for export. Their priorities are computed; element 3, for example, has priority -2, since it has two local neighbors (-2), one neighbor in the importing processor (+2),

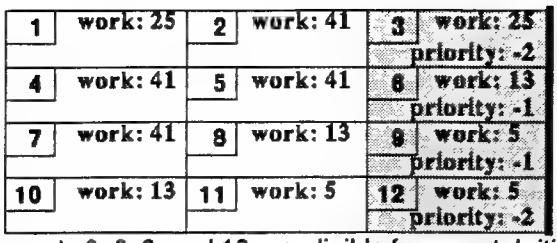
and one neighbor in some other processor (-2). Elements 6 and 9 share the highest priority, but element 6 is selected because it has a greater work load. Element 5 becomes eligible for export, but its priority is low since it has three local neighbors. The priorities are adjusted, and element 9 is selected, making element 8 a candidate. The priorities are again updated, and the selection process continues with elements 3 and 12 being selected. Although the work request is not completely satisfied, no other elements are exported, since the work loads of the elements with the highest priority, 5 and 8, are greater than the remaining work request.

EXAMPLE 3. We solve (3.7) with a fixed-order method ($p = 3$) on a 32×32-element mesh and tiling on 16 processors of the nCUBE/2 hypercube. In Figure 4.3 (left), we show the processor domain decomposition after 20 time steps. The tiling algorithm redistributes the work so that processors containing elements on the domain boundary have fewer elements than those in the interior of the domain. The global error of the numerical solution is 4.76766×10^{-3}. The total processing time was reduced by 5.18% from 128.86 seconds to 122.18 seconds by balancing once each time step. The average/maximum processor work ratio without balancing is 0.858; with balancing, it is 0.942. Parallel efficiency is increased from 90.80% without balancing to 95.58% with tiling.

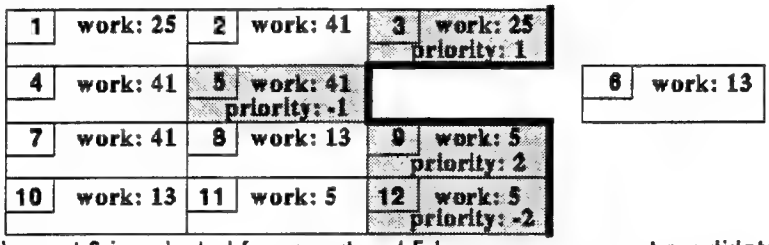
We then solve (3.7) using the adaptive p-refinement method on a 32×32-element mesh with TOL $= 3.5 \times 10^{-5}$ and tiling on 16 processors. In Figure 4.3 (right), we show the processor domain decomposition after 20 time steps. The shaded elements have higher-degree approximations and, thus, higher work loads. The tiling algorithm redistributes the work so that processors with high-order elements have fewer elements than those processors with low-order elements. The global error of the adaptive solution is 4.44426×10^{-3}, less than the fixed-order method above. The total processing time for the adaptive method was reduced 41.98% from 63.94 seconds to 37.10 seconds by balancing once each time step. The average/maximum processor work ratio without balancing is 0.362, and with balancing, it is 0.695. Parallel efficiency is increased from 35.10% without balancing to 60.51% with tiling.

EXAMPLE 4. We solve (3.7) for 225 time steps on all 1024 processors of the nCUBE/2 without balancing and with balancing once each time step. A fixed-order method with $p = 2$ produces a solution with global error 6.40644×10^{-2}. Using the tiling algorithm reduced the total execution time 6.25% from 1601.96 seconds without balancing to 1501.90 seconds with balancing (see Table 4.1). Parallel efficiency without balancing was 82.7%; with balancing, it was 88.2%.

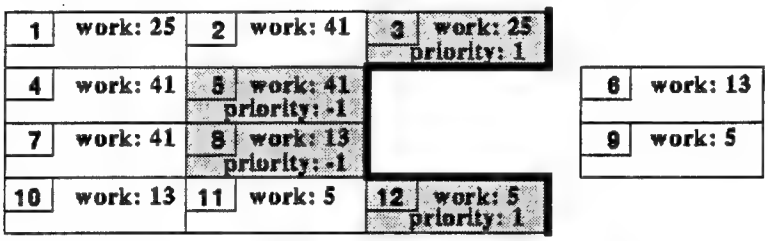
The adaptive p-refinement method produced a solution with global error 6.32205×10^{-2}, comparable to the fixed-order solution. With balancing, the maximum computation time (not including communication or balancing time) was reduced by 49.8% (see Table 4.1). The irregular subdomain boundaries created by the tiling algorithm increased the average



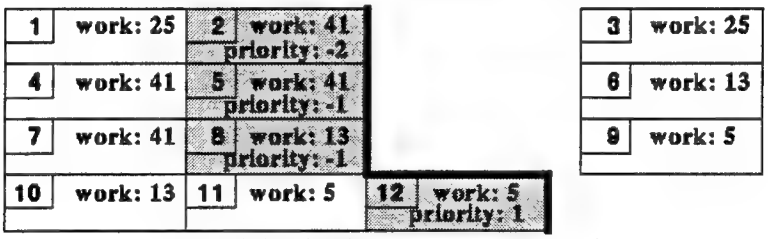
Elements 3, 6, 9, and 12 are eligible for export. Initial work request = 55.



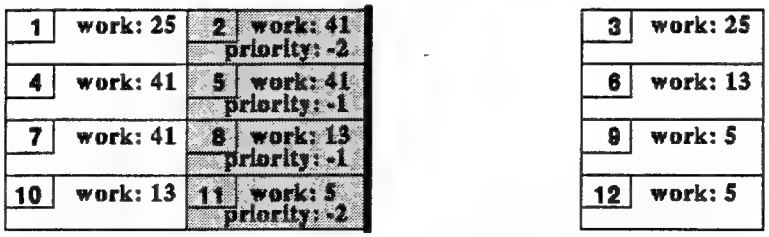
Element 6 is selected for export and 5 becomes an export candidate.
Work request = 42.



After second selection, work request = 37.



After third selection, work request = 12.



After fourth selection, work request = 7; no other elements are exported.

FIG. 4.2. *Example of element priorities and the export element selection algorithm.*

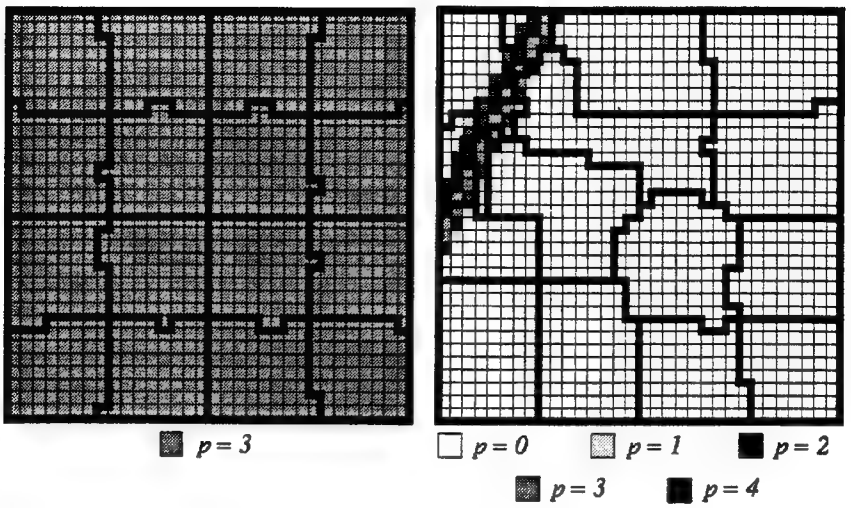


FIG. 4.3. *Processor domain decompositions after 20 time steps for Example 3 using fixed-order (left) and adaptive order (right) methods. Dark lines represent processor subdomain boundaries.*

communication time by 2.5%. Despite the extra communication time and the load balancing time, however, we see a 36.3% improvement in the total execution time.

In Figure 4.4, we show the maximum processing costs per time step, including the computation and balancing times, for the adaptive p-refinement method. The dashed and solid lines represent the maximum cost without and with balancing, respectively. The balanced computation's maximum cost per time step is significantly lower than without balancing. The spikes in both curves occur when the error tolerance was not satisfied on some elements and the adaptive p-refinement method back-tracked to compute a more accurate solution. In Figure 4.5, we show the cumulative maximum processing times with and without balancing. The immediate and sustained improvement of the application's performance is shown.

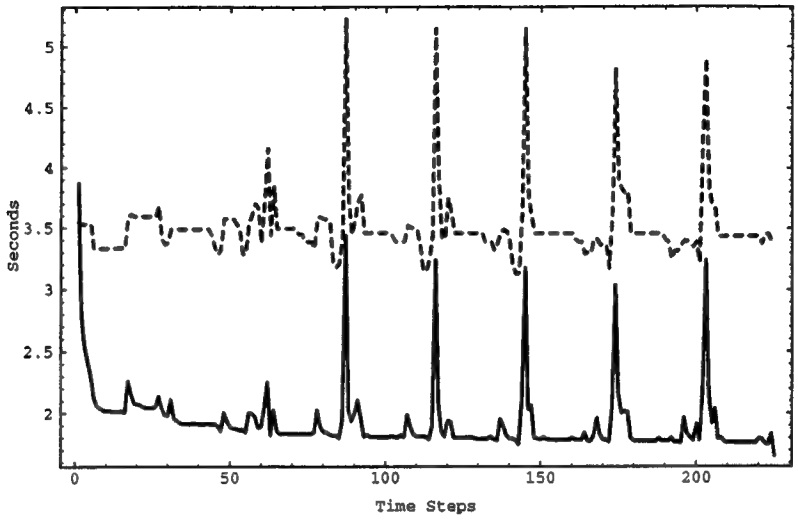


FIG. 4.4. *Maximum work load during each time step for Example 4 with (solid line) and without (dashed line) balancing.*

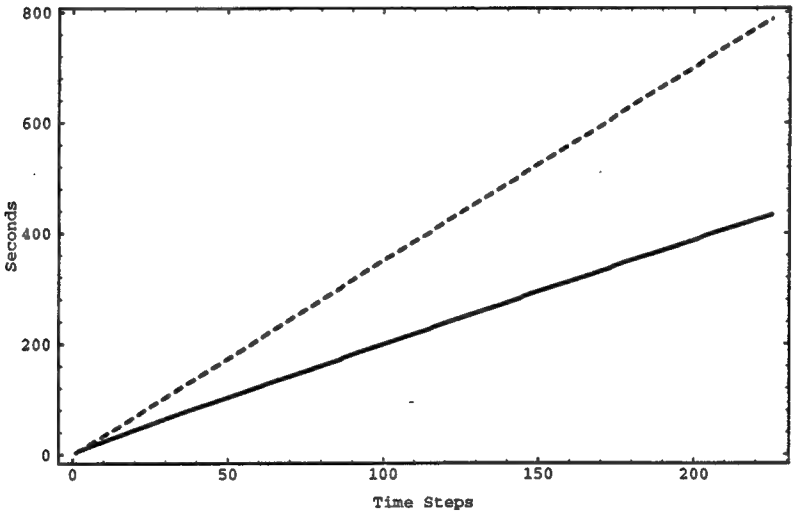


FIG. 4.5. *Cumulative maximum loads for Example 4 with (solid line) and without (dashed line) balancing.*

	Fixed-Order (p=2) Global Error: 0.06406		Adaptive p-refinement Global Error: 0.06322	
	Without Tiling	With Tiling	Without Tiling	With Tiling
Total Execution Time (seconds)	1601.96	1501.90	858.50	546.75
Max. Computation Time (seconds)	1549.77	1429.24	782.93	393.32
Average/Maximum Work Ratio	0.855	0.927	0.427	0.851
Avg. Communication Time (seconds)	59.09	59.09	70.85	72.65
Max. Balancing Time (seconds)	0.0	20.88	0.0	23.46
Parallel Efficiency	82.7%	88.2%	38.98%	61.21%

TABLE 4.1

Performance comparison for Example 4 using fixed-order and adaptive methods without and with balancing at each time step.

5. Three-Dimensional Mesh Partitioning. We describe a tree-based partitioning technique which utilizes the hierarchical structure of octree-derived unstructured meshes to distribute elemental data across processors' memories while reducing the amount of data that must be exchanged between processors. An octree-based mesh generator [30] recursively subdivides an embedding of the problem domain in a cubic universe into eight octants wherever more resolution is required. Octant bisection is initially based on geometric features of the domain but solution-based criteria are introduced during an adaptive h-refinement process. Finite element meshes of tetrahedral elements are generated from the octree by subdividing terminal octants.

In Figure 5.1, we illustrate the tree and mesh for a two-dimensional flow domain containing a small object. The root of the tree represents the entire domain (Figure 5.1c). The domain is recursively quartered until an adequate resolution of the object is obtained (Figure 5.1a). A smooth gradation is maintained by enforcing a one-level maximum difference between adjacent quadrants. After appropriate resolution is obtained, leaf quadrants are subdivided into triangular elements that are pointed to by leaf nodes of the tree (Figures 5.1b,c). The leaf quadrant containing the object must be decomposed into triangles based on the geometry of the object boundary. Smoothing, which normally follows element creation, is not shown.

Our tree-based based partitioning algorithm creates a one-dimensional ordering of the octree and divides it into nearly equal-sized segments based

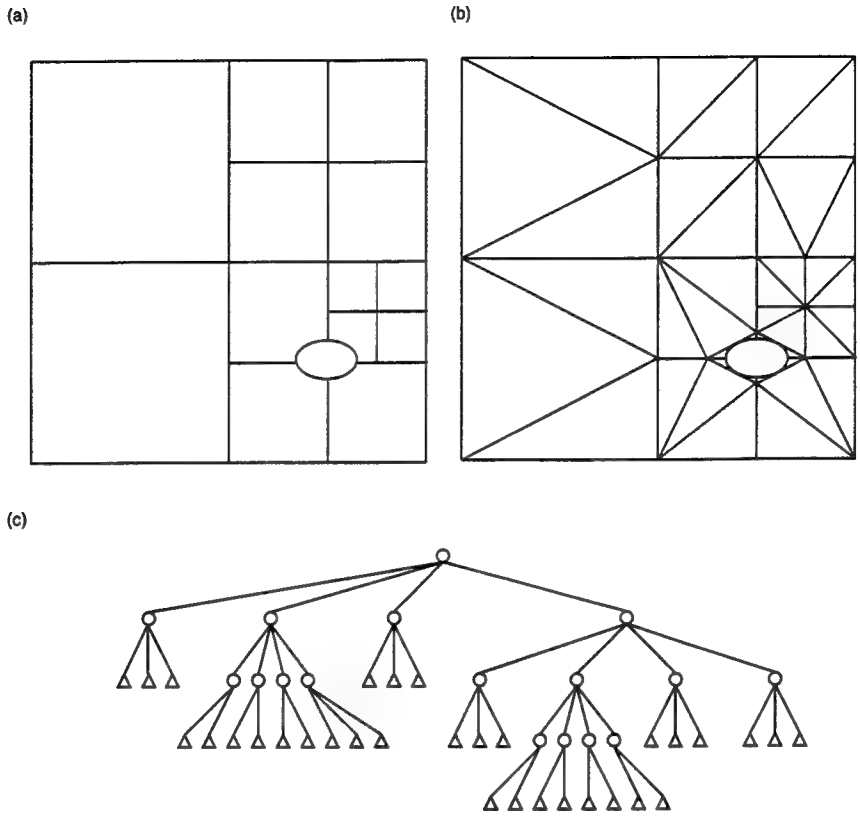


FIG. 5.1. *A quadtree representation of the flow field surrounding an object (a), division of terminal quadrants into triangular elements (b), and quadtree structure (c).*

on tree topology. The first step of the algorithm is the determination of cost metrics for all subtrees. Cost is currently defined as the number of elements within a subtree. For a leaf octant, this would simply be the number of tetrahedra associated with it. P-refinement would necessitate the inclusion of an element's order into the cost function. If the solution algorithm employs spatially-dependent time steps then, typically, a greater number of smaller time steps must be taken on smaller elements and this must also be reflected in the subtree cost. In any event, appropriate costs may be determined by a postorder traversal of the octree.

The second phase of the partitioning algorithm uses the cost information to construct the actual partitions. Since the number of partitions is prescribed and the total cost is known from the first phase, we also know the optimal size of each partition. Partitions consist of a set of octants that are each the root of a subtree and are determined by a truncated depth-first search. Thus, octree nodes are visited in depth-first order, and

subtrees are accumulated into successive partitions. The subtree rooted at the visited node is added to the current partition if it fits. If it would exceed the optimal size of the current partition, a decision must be made as to whether it should be added, or whether the traversal should examine it further. In the latter case, the traversal continues with the offspring of the node and the subtree may be divided among two or more partitions. The decision on whether to add the subtree or examine it further is based on the amount by which the optimal partition size is exceeded. A small excess may not justify an extensive search and may be used to balance some other partition which is slightly undersized. When the excess at a node is too large to justify inclusion in the current partition, and the node is either terminal or sufficiently deep in the tree, the partition is closed and subsequent nodes are added to the next partition.

This partitioning method requires storage for nonterminal nodes of the tree which would normally not be necessary since they contain no solution data. However, only minimal storage costs are incurred since information is only required for tree connectivity and the cost metric. For this modest investment, we have a partitioning algorithm that only requires $O(J)$ serial steps.

Partitions formed by this procedure do not necessarily form a single connected component; however, the octree decomposition and the orderly tree traversal tend to group neighboring subtrees together. Furthermore, a single connected component is added to the partition whenever a subtree fits within the partition.

A tree-partitioning example is illustrated in Figure 5.2. All subtree costs are determined by a post order traversal of the tree. The partition creation traversal starts at the root, Node 0 (Figure 5.2a). The node currently under investigation is identified by a double circle. The cost of the root exceeds the optimal partition cost, so the traversal descends to Node 1 (Figure 5.2b). As shown, the cost of the subtree rooted at node 1 is smaller than the optimal partition size and, hence, this subtree is added to the current partition, p0, and the traversal continues at Node 2 (Figure 5.2c). The cost of the subtree rooted at Node 2 is too large to add to p0, so the algorithm descends to an offspring of Node 2 (Figure 5.2d). Assuming Node 4 fits in p0, the traversal continues with the next offspring of Node 2 (Figure 5.2e). Node 5 is a terminal node whose cost is larger than the available space in p0, so the decision is made to close p0 and begin a new partition, p1. As shown (Figure 5.2f), Node 5 is very expensive, and when the traversal is continued at Node 3, p1 must be closed and work continues with partition p2.

The tree-traversal partitioning algorithm may easily be extended for use with a parallel adaptive environment. An initial partitioning is made using the serial algorithm described above. As the numerical solution advances in time, h- and/or p-refinement introduces a load imbalance. To obtain a new partitioning, let each processor compute its subtree costs us-

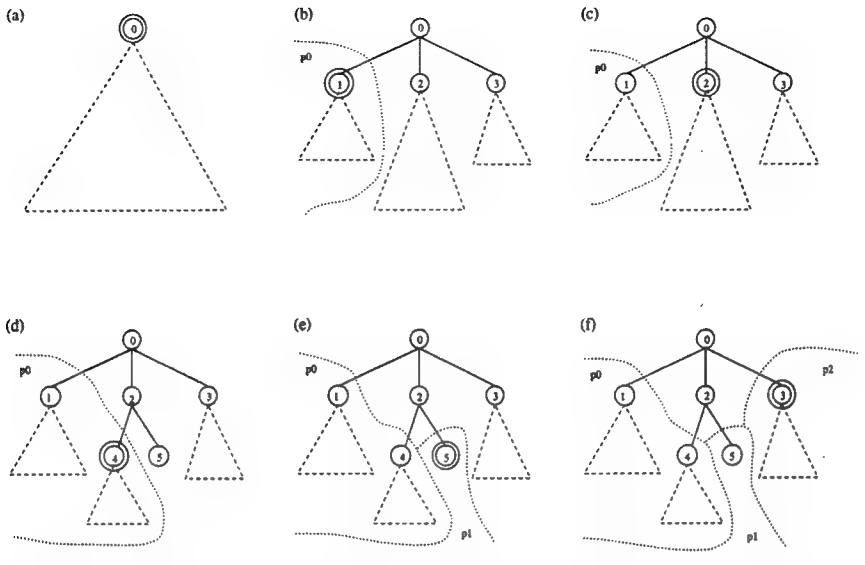


FIG. 5.2. *A tree partitioning example. The partition-creation traversal starts at the root (a). Nodes are visited and added to the current partition if their subtree fits (b). When a subtree is too large to fit (c), the traversal descends into the subtree (d). Alternatively, the partition is closed and work begins on a new partition (e). The process continues until the traversal is complete (f).*

ing the serial traversal algorithm within its domain. This step requires no interprocessor communication. An inexpensive parallel prefix operation may be performed on the processor-subtree totals to obtain a global cost structure. This information enables a processor to determine where its local tree traversal is located in the global traversal.

Now, following the serial procedure, each processor may traverse its subtrees to create partitions. A processor determines the partition number to start working on based on the total cost of processors preceeding it. Each processor starts counting with this prefix cost and traverses its subtrees adding the cost of each visited node to this value. Partitions end near cost mutiples of N/P, where N is the total cost and P is the number of processors. Exceeding a multiple of N/P during the traversal is analagous to exceeding the optimal partition size in the serial case and the same criteria may be used to determine where to end partitions. When all processors finish their traversals, each subtree (and its associated data) is assigned to a new partition and may be migrated to its new location. Migration may be done using global communication; however, on some architectures, it may be more efficient to move data via simultaneous processor shift operations. This linear communication pattern is made possible by the one-dimensional nature of the partition traversal.

While the cost of computing the new partition is small, the cost of

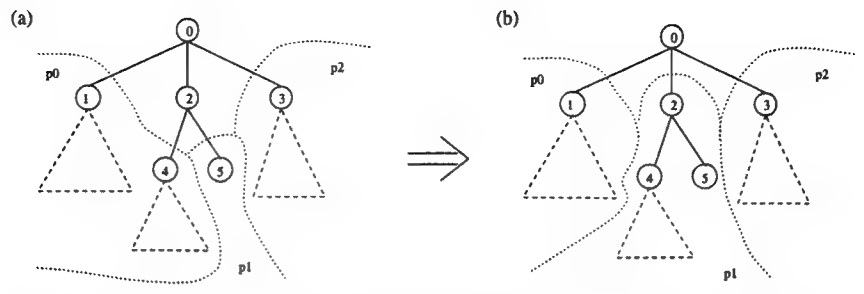


FIG. 5.3. *Iterative rebalancing of tree-based partitions. The subtree rooted at Node 4 (a) has been shifted from p0 to p1 (b) to relieve a load imbalance. The new root of p1 is Node 2, the common parent of Nodes 4 and 5.*

data movement is likely to be high and it would be desirable to amortize this by tolerating small imbalances. A strategy to delay the need for complete repartitioning would simply shift partition boundaries, thus, migrating subtrees from a processor P_n to its neighbors P_{n-1} and P_{n+1}. If, for example, processor P_n seeks to transfer cost m to P_{n-1}, it simply traverses its subtrees accumulating their costs until it reaches m. The nodes visited comprise a subtree which may be transferred to P_{n-1} and which is contiguous with the subtrees in P_{n-1}. Likewise, if P_n desires to transfer work to P_{n+1}, the reverse traversal could remove a subtree from the trailing part of P_n. Consider, as an example, the subtree rooted at Node 4 of Figure 5.3a and suppose that its cost has increased through refinement. In Figure 5.3b, we show how the partition boundary may be shifted to move the subtree rooted at Node 4 to partition p1. The amount of data to be moved from processor to processor may utilize a relaxation algorithm or the tiling procedure discussed in Section 4.

EXAMPLE 5. Performance results obtained by applying the tree-based mesh partitioning algorithm to various three-dimensional irregular meshes are presented in Figure 5.4. The meshes were generated by the Finite Octree mesh generator [30]. "Airplane" is a 182K-element mesh of the volume surrounding a simple airplane [17]. "Copter" is a 242K-element mesh of the body of a helicopter [17]. "Onera," "Onera2," and "Onera3" are 16K-, 70K-, and 293K-element meshes, respectively, of the space surrounding a swept, untwisted Onera-M6 wing which has been refined to resolve a bow shock [18]. "Cone" is a 139K-element mesh of the space around a cone having a 10° half-angle and which also has been refined to resolve a shock.

The quality of a partition has been measured as the percent of element faces lying on inter-partition boundaries relative to the total number of faces of the mesh. Graphs in Figure 5.4 display these percentages as a function of the optimal partition size. In all cases the cost variance between the partitions is very small (about as small as the maximum cost of a leaf octant). The proportion in Figure 5.4 is, in a sense, the total surface area

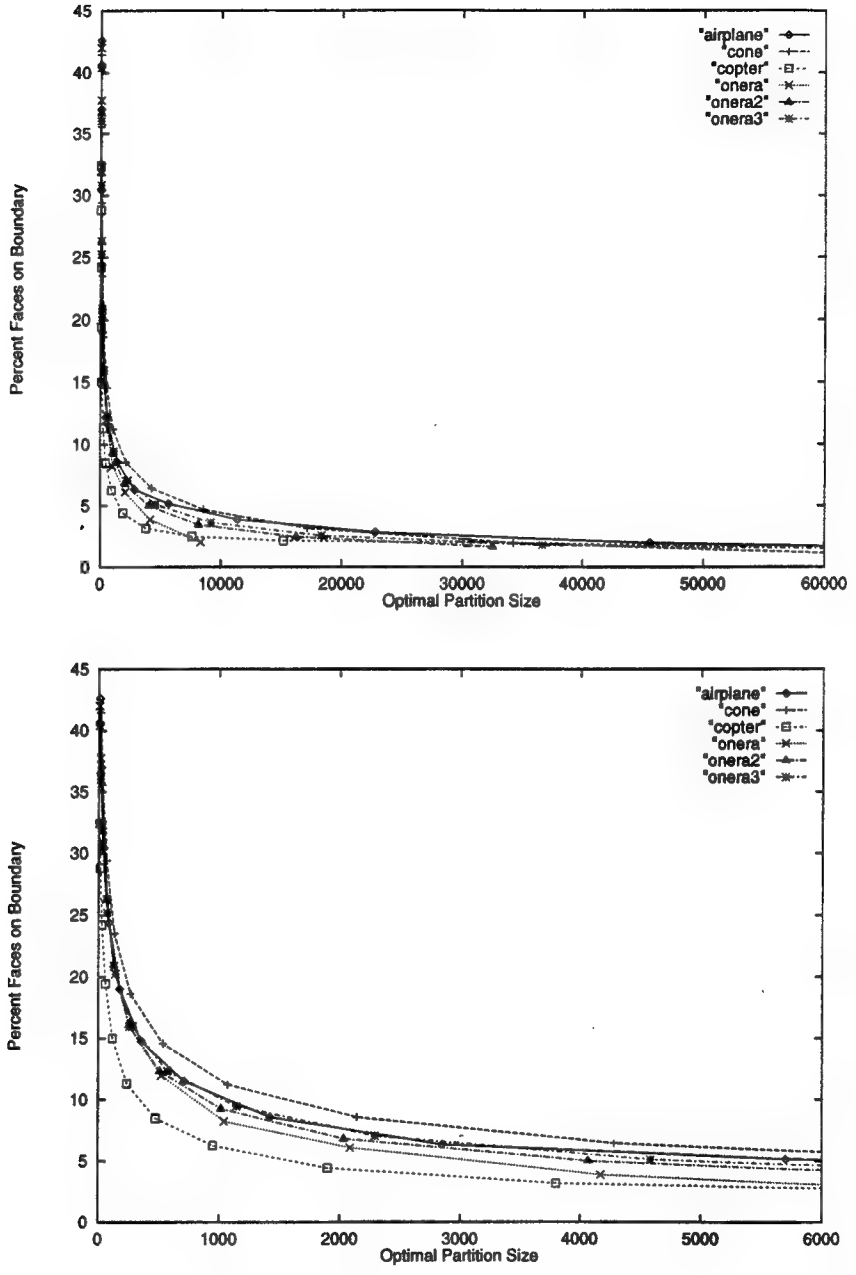


FIG. 5.4. *Performance of the tree partitioning algorithm on five meshes: large-scale (top) and small-scale (bottom).*

that partitions hold in common. Smaller ratios require less communication relative to the amount of local data access. This measure is closely related to the number of "cuts" that the partition creates [23,20,32]; however, we have chosen to normalize by the total number of faces in order to compare partition quality over a wide range of mesh sizes and number of partitions.

In large scale (top) the data of Figure 5.4 show the expected behaviour that the interface proportion approaches zero as the partition size increases (due to the number of partitions approaching unity). Conversely, as the optimal partition size approaches unity (due to number of partitions approaching the number of elements), the interface proportion goes to unity. Examination of the small scale (bottom) results reveals that the interface proportion is less than 12% when the partition size exceeds 1000 for these meshes. Interfaces drop to below 9% and 8%, respectively, for partition sizes of 2000 and 3000. This performance is comparable to recursive spectral bisection [22] but requires much less computation ($O(J)$ as opposed to $O(J^2)$ [27]).

The best performance occurred with the helicopter mesh, which was the only mesh of a solid object (as opposed to a flow field surrounding an object). The solid can easily be cut along its major axis to produce partitions with small inter-partition boundaries, and was included for generality. The lowest performance occurred with the cone mesh. This is most likely due to the model and shock region being conically shaped, which is somewhat at odds with the rectangular decomposition imposed by the octree.

In general, inter-partition boundaries should be less than 10%, indicating partition sizes of 2000 or more. This minimum partition size is not an excessive constraint, since a typical three-dimensional problem employing a two million-element mesh being solved on a 1024-processor computer would have about 2000 elements per processing element.

Another measure of partition quality is the percent of a partition's element faces lying on inter-partition boundaries relative to the total number of faces in that partition. This number is, in a sense, the ratio of surface area to volume of a partition. For our example meshes, this measure was below 22% and 18%, respectively, for partition sizes of 1000 and 1500.

EXAMPLE 6. In Figure 5.5 we show partitions of several meshes from Example 5. The partitions exhibit a blocked structure; however, several partitions of the airplane mesh appear to be made up of disconnected components. While this is possible, although unlikely, in this case the partitions appear to be disconnected because the display is a two-dimensional slice through the three-dimensional domain.

EXAMPLE 7. In Figure 5.6 we show the pressure contours of a Mach 2 Euler flow (1.1,2.6) past the "Cone" mesh of Example 5. The solution employs van Leer's flux vector splitting (2.7) and was computed on a Thinking Machines CM-5 with 128 processors. Several iterations of h-refinement were required to yield this mesh. At each iteration, elements were marked with

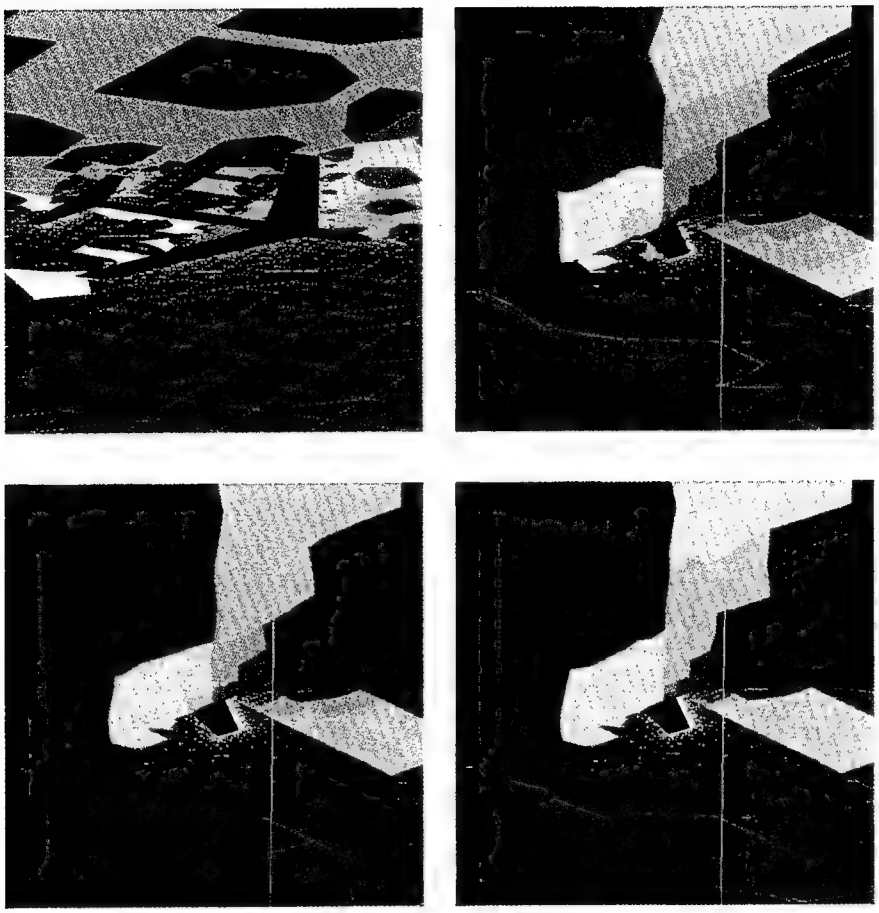


FIG. 5.5. *The airplane mesh, and three refinements of the Onera M6 wing mesh, all divided into 32 partitions. Each color represents a different partition.*

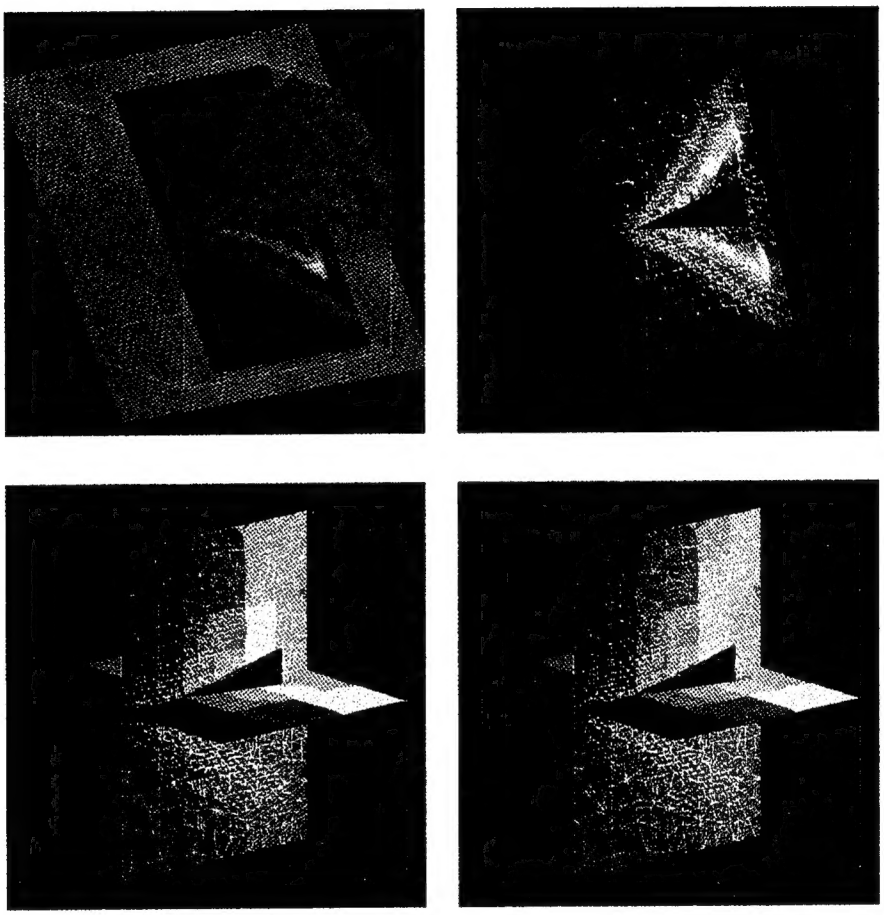


FIG. 5.6. *Shock surface and pressure contours found when computing the Mach 2 flow past a cone having a half-angle of* 10° *(top). Partitions of the mesh into 16 (left) and 32 (right) pieces (bottom). Each color represents a different partition.*

the desired tree level (either larger for refinement, or smaller for coarsening), and a new global mesh created to satisfy these constraints. The shock surface and pressure contours are shown above; below are examples of how the mesh may be partitioned for 16 and 32 processor machines. Each color represents membership in a different partition (and, hence, residence on a different processor).

6. Conclusion. We have demonstrated the effectiveness of adaptive methods for solving systems of hyperbolic conservation laws on massively parallel computers. Using a discontinuous Galerkin finite element method with projection limiting of moments of the solution within an element, we can model problems with discontinuities sharply without spurious oscillations. The discontinuous Galerkin method has a small computational stencil, enabling its efficient implementation on massively parallel computers. Adaptive p- and h-refinement methods provide faster convergence than traditional methods, but their nonuniform work loads create load imbalance on parallel computers, reducing the parallel efficiency of the methods. We correct the load imbalance by using a dynamic load balancing technique called tiling that produces a global balance by performing local balancing within overlapping neighborhoods of processors. Using tiling and adaptive p-refinement, computation of a two-dimensional example required approximately one-third as much time as a fixed-order computation with the same global accuracy. In three dimensions, we have demonstrated the effectiveness of a tree-based mesh partitioning algorithm for reducing parallel communication costs. This algorithm performs almost as well as recursive spectral bisection, but requires much less work to compute a partitioning.

In future work, we will combine the adaptive h- and p-refinement techniques to obtain an adaptive hp-refinement method that can optimize computational effort in both smooth and discontinuous solution regions. We will extend the tiling algorithm to incorporate the changing data structures required for h-refinement, and experiment with load balancing strategies for adaptive hp-refinement meshes. The tree-based partitioning algorithm will be extended to operate in parallel, and we will experiment with dynamic rebalancing strategies.

7. Acknowledgements. We wish to thank Thinking Machines Corporation, and in particular Zdeněk Johan and Kapil Mathur, for their assistance with the CM-5.

REFERENCES

[1] S. Adjerid, M. Aiffa, and J. E. Flaherty, *Adaptive Finite Element Methods for Singularly Perturbed Elliptic and Parabolic Systems*, submitted for publication, 1993.

[2] S. Adjerid and J. E. Flaherty, *Second-Order Finite Element Approximations and A Posteriori Error Estimation for Two-Dimensional Parabolic Systems*, Numer. Math., Vol. 53, 1988, pp. 183–198.

[3] S. ADJERID, J. FLAHERTY, P. MOORE, AND Y. WANG, *High-Order Adaptive Methods for Parabolic Systems*, Physica-D, Vol. 60, 1992, pp. 94–111.

[4] S. ADJERID, J. FLAHERTY, AND Y. WANG, *A Posteriori Error Estimation with Finite Element Methods of Lines for One-Dimensional Parabolic Systems*, Numer. Math., Vol. 65, 1993, pp. 1–21.

[5] D. C. ARNEY AND J. E. FLAHERTY, *An Adaptive Mesh Moving and Local Refinement Method for Time-Dependent Partial Differential Equations*, ACM Trans. Math. Software, Vol. 16, 1990, pp. 48–71.

[6] I. BABUSKA, *The p- and hp-Versions of the Finite Element Method. The State of the Art*, in "Finite Elements: Theory and Appliations," Springer-Verlag, New York, 1988.

[7] I. BABUSKA, B. A. SZABO, AND I. N. KATZ, *The p-Version of The Finite Element Method*, SIAM J. Numer. Anal., Vol. 18, 1981, pp. 515–545.

[8] M. J. BERGER AND J. OLIGER, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, J. Comput. Phys., Vol. 53, 1984, pp. 484–512.

[9] K. S. BEY AND J. T. ODEN, *An A Posteriori Error Estimate for Hyperbolic Conservation Laws*, preprint, 1993.

[10] M. B. BIETERMAN AND I. BABUSKA, *The Finite Element Method for Parabolic Equations, II. A Posteriori Error Estimation and Adaptive Approach*, Numer. Math., Vol. 40, 1982, pp. 373–406.

[11] R. BISWAS, K. D. DEVINE, AND J. E. FLAHERTY, *Parallel, Adaptive Finite Element Methods for Conservation Laws*, Appl. Numer. Math., 1993, to appear.

[12] R. BISWAS, J. E. FLAHERTY, AND D. C. ARNEY, *An Adaptive Mesh Moving and Refinement Procedure for One-Dimensional Conservation Laws*, Appl. Numer. Math., 1993, to appear.

[13] B. COCKBURN, S.-Y. LIN, AND C.-W. SHU, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One-Dimensional Systems*, J. Comput. Phys., Vol. 84, 1989, pp. 90–113.

[14] B. COCKBURN AND C.-W. SHU, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework*, Math. Comp., Vol. 52, 1989, pp. 411–435.

[15] B. COCKBURN, S.-Y. LIN, AND C.-W. SHU, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case*, Math. Comp., Vol. 54, 1990, pp. 545–581.

[16] P. DEVLOO, J. T. ODEN, AND P. PATTANI, *An h-p Adaptive Finite Element Method for the Numerical Simulation of Compressible Flow*, Comput. Methods Appl. Mech. Engng., Vol. 70, 1988, pp. 203–235.

[17] S. DEY, *personal communication*, 1993.

[18] M. DINAR, *personal communication*, 1993.

[19] S. HAMMOND, *Mapping Unstructured Grid Computations to Massively Parallel Computers*, Ph.D. Dissertation, Rensselaer Polytechnic Institute, Dept. Comp. Sci., Troy, 1992.

[20] B. HENDRICKSON AND R. LELAND, *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*, Sandia National Laboratories Tech. Rep. SAND92-1460, Albuquerque, 1992.

[21] B. HENDRICKSON AND R. LELAND, *Multidimensional Spectral Load Balancing*, Sandia National Laboratories Tech. Rep. SAND93-0074, Albuquerque, 1993.

[22] Z. JOHAN, *personal communication*, 1993.

[23] Z. JOHAN, K. MATHUR, AND S. L. JOHNSSON, *An Efficient Communication Strategy for Finite Element Methods on the Connection Machine CM-5 System*, Thinking Machines Tech. Rep. No. 256, 1993, submitted for publication.

[24] E. LEISS AND H. REDDY, *Distributed Load Balancing: Design and Performance Analysis*, W. M. Keck Research Computation Laboratory, Vol. 5, 1989, pp. 205–270.

[25] R. A. LUDWIG, J. E. FLAHERTY, F. GUERINONI, P. L. BAEHMANN, AND M. S. SHEPHARD, *Adaptive Solutions of the Euler Equations Using Finite Quadtree*

and Octree Grids, Computers and Structures, Vol. 30, 1988, pp. 327–336.

[26] P. K. MOORE AND J. E. FLAHERTY, *A Local Refinement Finite-Element Method for One-Dimensional Parabolic Systems*, SIAM J. Numer. Anal., Vol. 27, 1990, pp. 1422–1444.

[27] A. POTHEN, H. SIMON, AND K.-P. LIOU, *Partitioning Sparse Matrices with Eigenvectors of Graphs*, SIAM J. Matrix Analysis and Applications, Vol. 11, 1990, pp. 430–452.

[28] E. RANK AND I. BABUSKA, *An Expert System for the Optimal Mesh Design in the hp-Version of the Finite Element Method*, Int. J. Numer. Meths. Engng., Vol. 24, 1987, pp. 2087–2106.

[29] H. N. REDDY, *On Load Balancing*, Ph.D. Dissertation, Dept. Comp. Sci., Univ. of Houston, Houston, TX, 1989.

[30] M. S. SHEPHARD AND M. K. GEORGES, *Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique*, Int. J. Numer. Meths. Engng., Vol. 32, No. 4, 1991, pp. 709-749.

[31] C.-W. SHU AND S. OSHER, *Efficient Implementation of Essentially Non-Oscillatory Shock-Capturing Schemes, II*, J. of Comput. Phys., Vol. 27, 1978, pp. 1–31.

[32] H. D. SIMON, *Partitioning of Unstructured Problems for Parallel Processing*, Comput. Systs. Engng., Vol. 2, 1991, pp. 135–148.

[33] P. K. SWEBY, *High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws*, SIAM J. Numer. Anal., Vol. 21, 1984, pp. 995–1011.

[34] B. SZABO AND I. BABUSKA, *Introduction to Finite Element Analysis*, J. Wiley and Sons, New York, 1990.

[35] B. VAN LEER, *Flux Vector Splitting for the Euler Equations*, ICASE Report. No. 82-30, Inst. Comp. Applics. Sci. Engng., NASA Langley Research Center, Hampton, 1982.

[36] B. VAN LEER, *Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection*, J. Comput. Phys., Vol. 23, 1977, pp. 276–299.

[37] S. R. WHEAT, *A Fine Grained Data Migration Approach to Application Load Balancing on MP MIMD Machines*, Ph.D. Dissertation, Dept. Comp. Sci., Univ. of New Mexico, Albuquerque, 1992.

[38] S. R. WHEAT, K. D. DEVINE, AND A. B. MACCABE, *Experience with Automatic, Dynamic Load Balancing and Adaptive Finite Element Computation*, Proc. Hawaii Int. Conf. System Sciences, 1994, to appear.